



Frederic Legrand presents:

Using Service Mesh for Resiliency and Observability

Au programme



Le Service Mesh, c'est quoi ?



Cas d'utilisation



Bénéfices de OpenShift Service Mesh



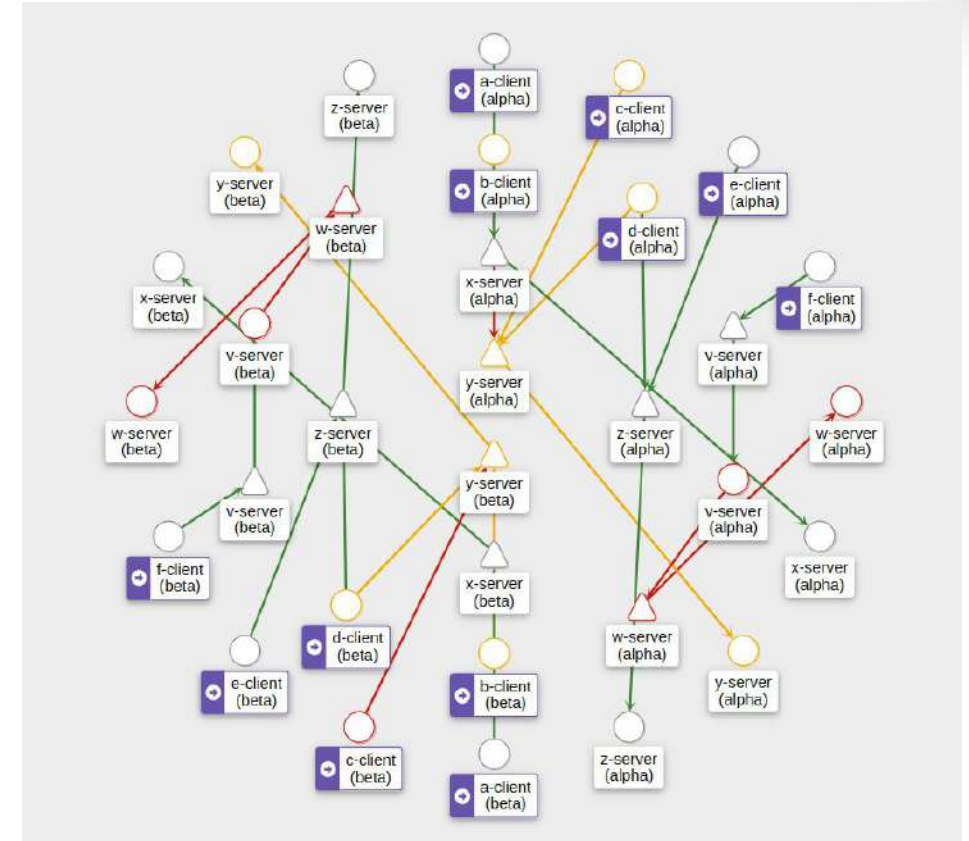
Démonstration

Le Service Mesh, c'est quoi ?

Les Micro Services

Les microservices sont des systèmes décentralisés
Le dépannage des systèmes décentralisé est difficile :

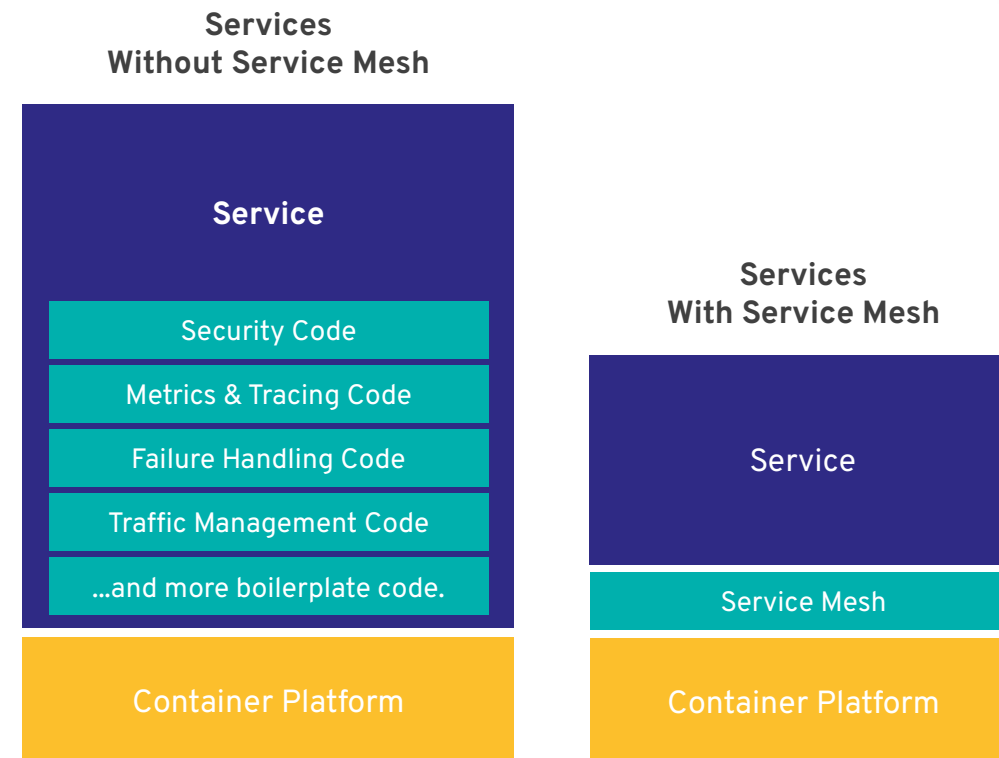
- Retards et échecs sporadiques
- Performances décevantes
- Découverte de failles de sécurité
- Services améliorés sans répercussion positive sur les performances
- Les correctifs apportés causent d'autres problèmes



Pourquoi utiliser le Service Mesh ?

Une solution pour les challenges qu'apportent les Micro Services :

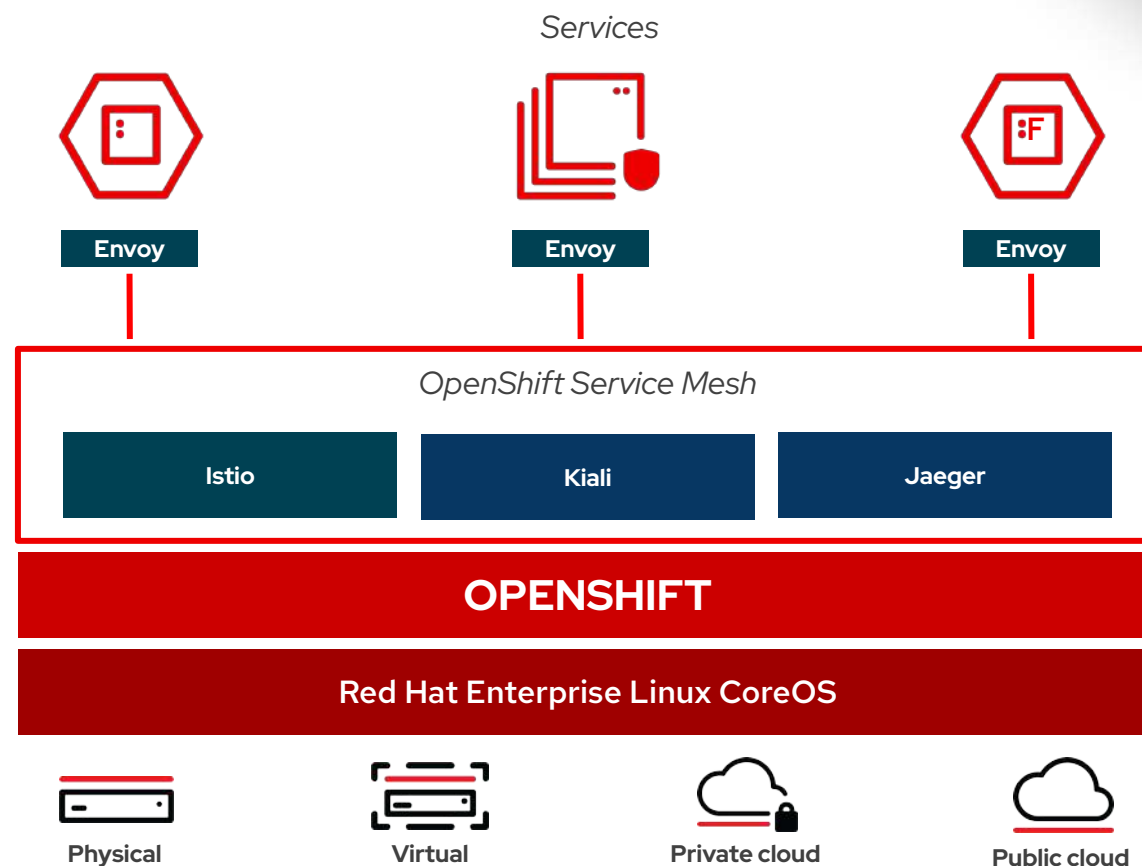
- Résout les problèmes des systèmes décentralisés avec une **couche d'infrastructure commune**
- **Réduit les erreurs** de copié/collé et de « boilerplate » code
- Permet d'appliquer des **politiques communes à tous les services**
- Fin des **cross-cutting concerns** pour les développeurs



OpenShift Service Mesh, quels avantages ?

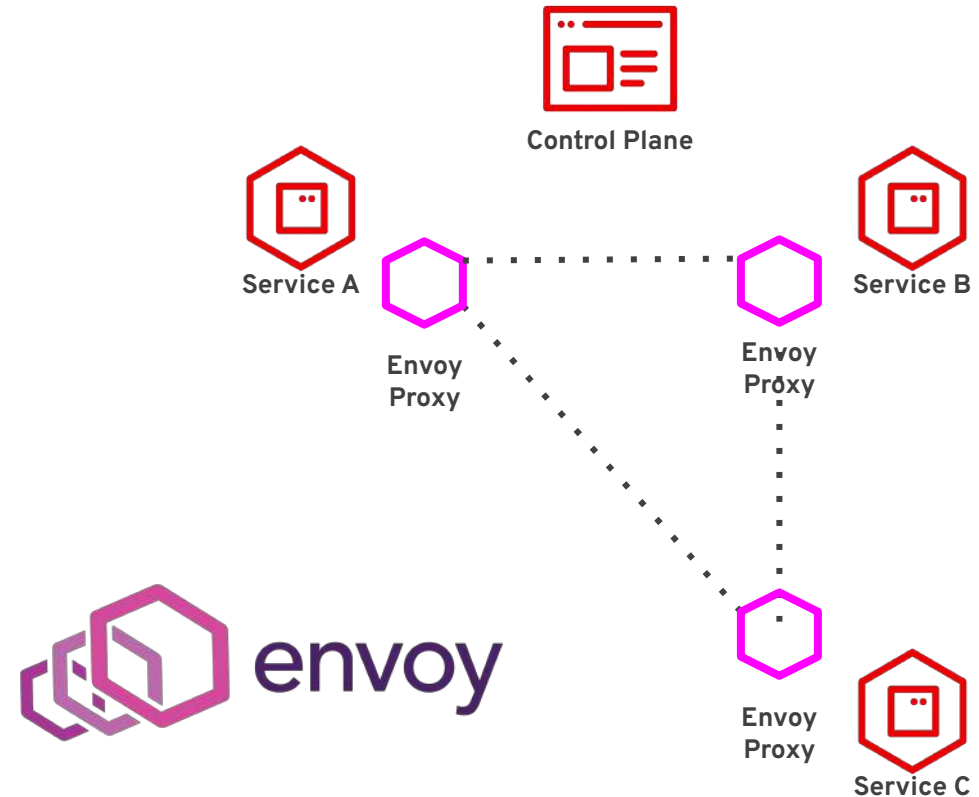
Connecter, sécuriser, contrôler et observer les services sur OpenShift

- Couche d'infrastructure logicielle entre **Kubernetes** et les services pour gérer les communications
- Gère les défis communs des "microservices", afin que les développeurs n'aient pas à le faire
 - Sécurité
 - Monitoring
 - Résilience des applications
 - Mises à niveau, déploiements et tests A/B
 - Etc.



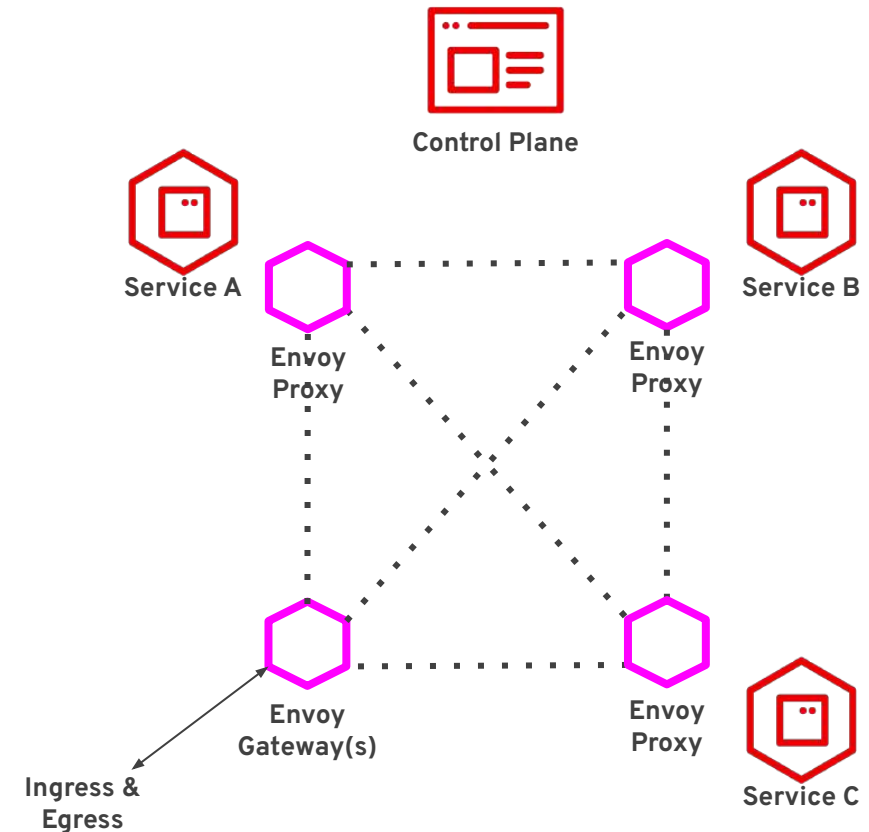
Relier les services entre eux au sein du Service Mesh - en Interne

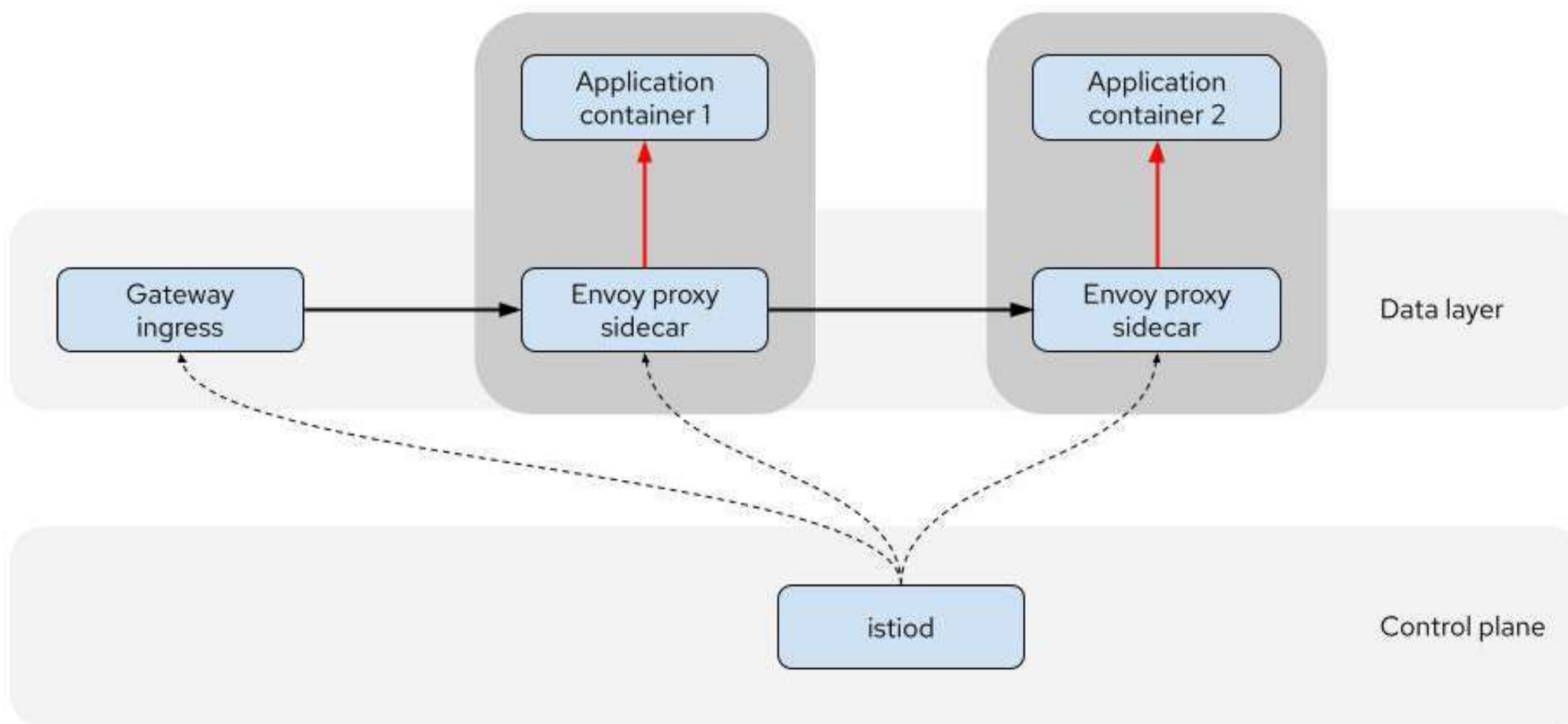
- Tous les pods de service reçoivent un **proxy Envoy** comme conteneur « sidecar ». Ensemble, ils forment le « **Data Plane** »
- Toutes les communications passent par ces **proxys**.
- Un maillage des communications est donc créé et donne une **visibilité et un contrôle total de tout le trafic**.
- Les proxys, et donc le Service Mesh, sont configurés et gérés par un « **Control Plane** » central



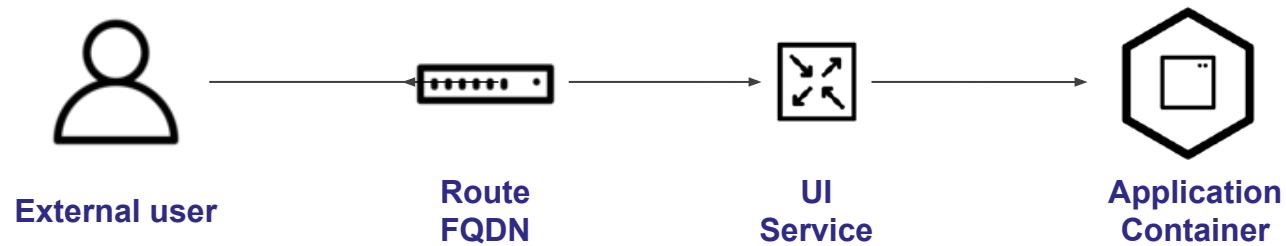
Relier les services à l'extérieur du Service Mesh - en Externe

- La communication externe se fait par l'intermédiaire de Gateways de proxys, qui font également partie du Service Mesh.
- Les passerelles **Ingress** gèrent le trafic **entrant**
- Les passerelles **Egress** gèrent le trafic **sortant**
- Sur OpenShift, les Service Mesh Ingress Gateways peuvent être utilisées conjointement avec une route OpenShift

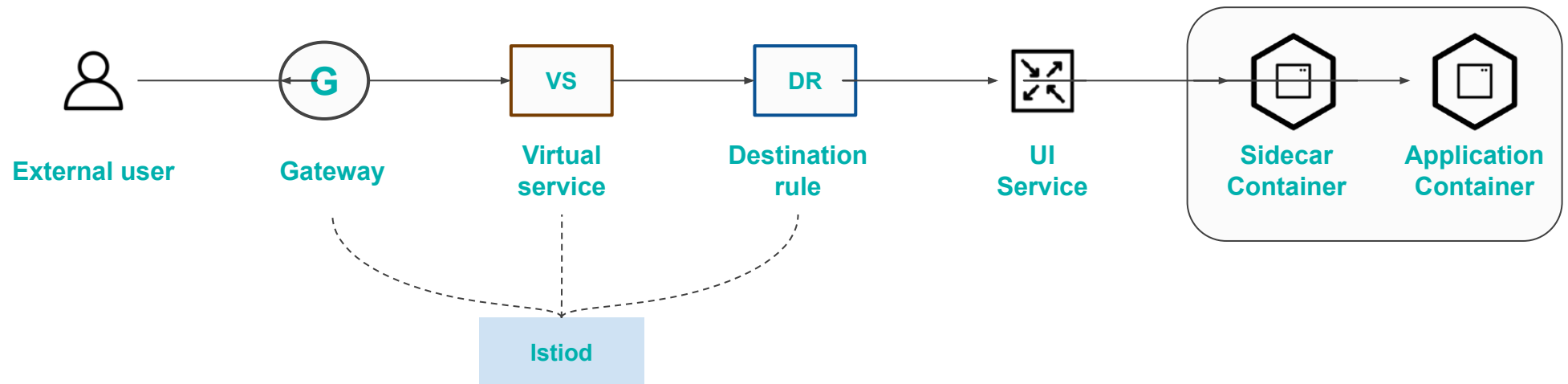




Without service mesh



With service mesh



OpenShift Service Mesh

Bénéfices

OpenShift Service Mesh



**Multi-Tenant
Architecture**

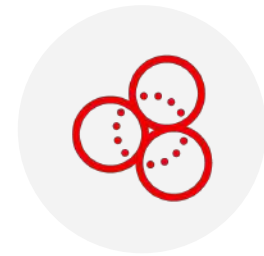


**Security & Compliance
Focus**



**Management, Monitoring
& Observability**

Pre-configured Kiali,
Jaeger and Grafana for
simplified management,
monitoring and
observability.

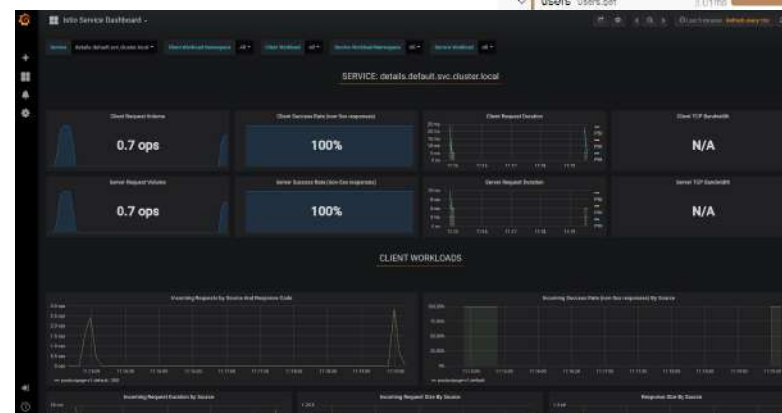
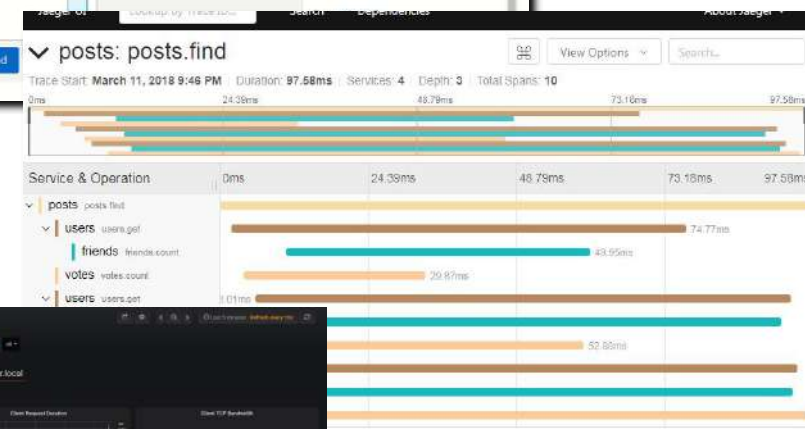
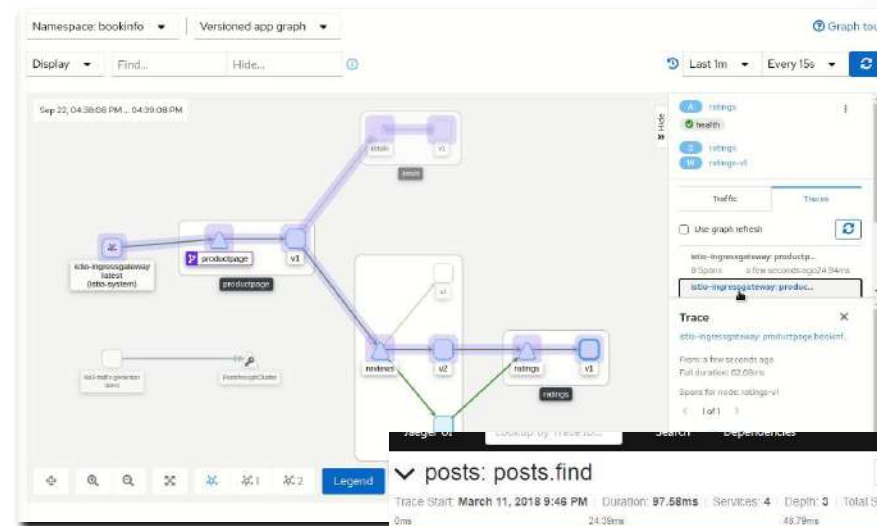


**OpenShift
Integrations**

Management, Monitoring & Observability

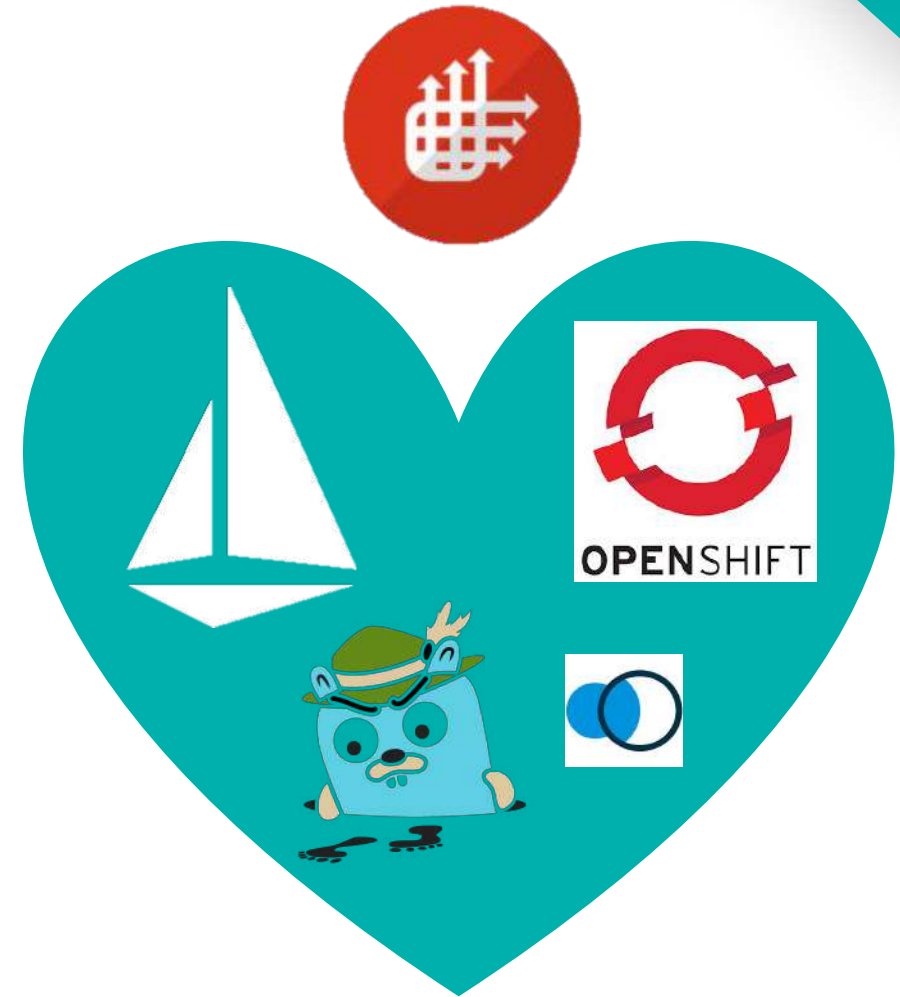
OpenShift Service Mesh comprend une pile intégrée pour la gestion, la surveillance et l'observabilité :

- Kiali, avec sa vue topologique, peut être utilisé pour observer, gérer et dépanner le Service Mesh.
- Grafana et Prometheus fournissent des mesures et un suivi prêts à l'emploi pour tous les services.
- Jaeger et Elasticsearch capturent les trames en fournissant une vue "par requête" pour isoler les goulots d'étranglement entre les services.



OpenShift / Istio un couple de choc !

- Upstream du projet Istio.io, downstream du projet Maistra.
- Red Hat effectue une validation et un contrôle qualité sur les versions Upstream d'Istio afin de s'assurer qu'elles sont prêtes à être mises en production.
- OpenShift Service Mesh 2.2 est basé sur Istio 1.12.
- Red Hat et IBM sont les plus grands contributeurs à Istio, derrière Google qui a créé le projet.

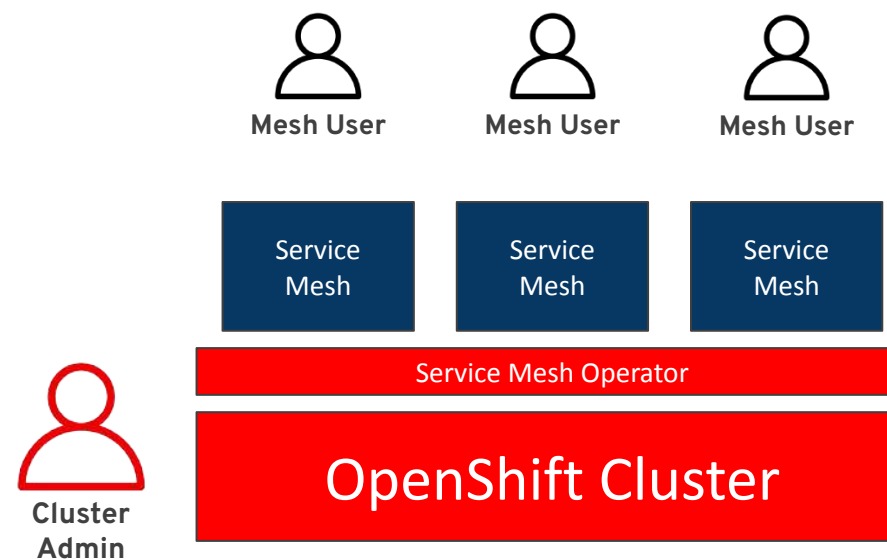


Sécurité et

Conformité

Permissions réduites pour l'administration de Service Mesh :

- Istio nécessite des droits élevés pour administrer le Service Mesh
- OpenShift, grâce au Service Mesh Operator, effectue des opérations privilégiées individuellement pour chaque Control Plane.
- Les composants du Service Mesh ont uniquement de la visibilité dans leur namespace.
- Cela permet une réduction du niveau de permission requis pour gérer le Service Mesh. Les droits sont gérés avec le RBAC Kubernetes



Service Mesh

Personas

Permissions "nécessaires" pour l'administration :

- **Cluster Admins :**
 - Gère les clusters et l'infrastructure.
 - Souvent un groupe central d'opérations/infra.
- **Administrateur(s) de Service Mesh :**
 - Gère un ou plusieurs Service Mesh(s) - connectivité des applications et sécurité au sein du Service Mesh.
 - Ne nécessite pas d'administrateur de cluster.
- **Administrateur(s) de services :**
 - Responsable d'un ou plusieurs services, mais peut ne pas gérer les ressources du Service Mesh.



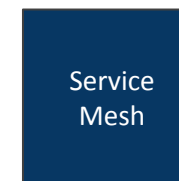
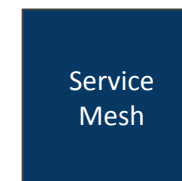
Service Admin(s)



Mesh Admin(s)



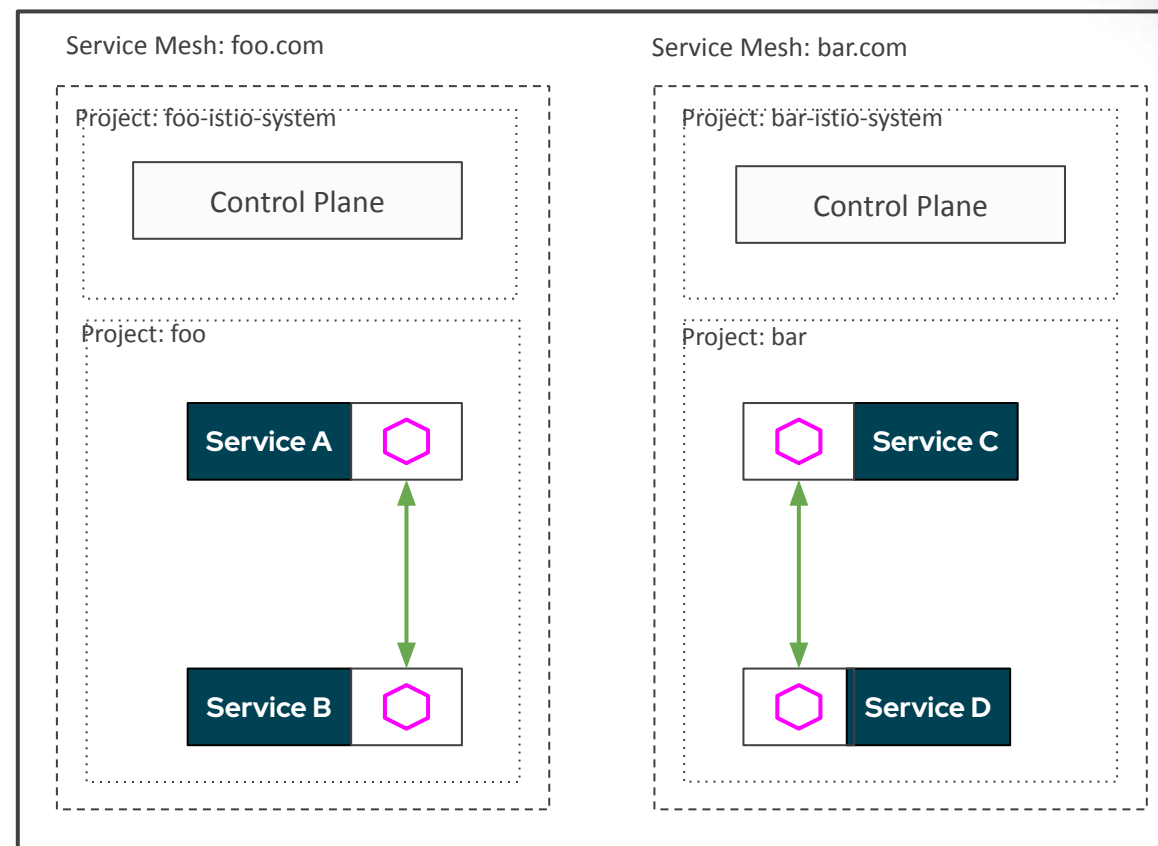
Cluster Admin(s)



Service Mesh

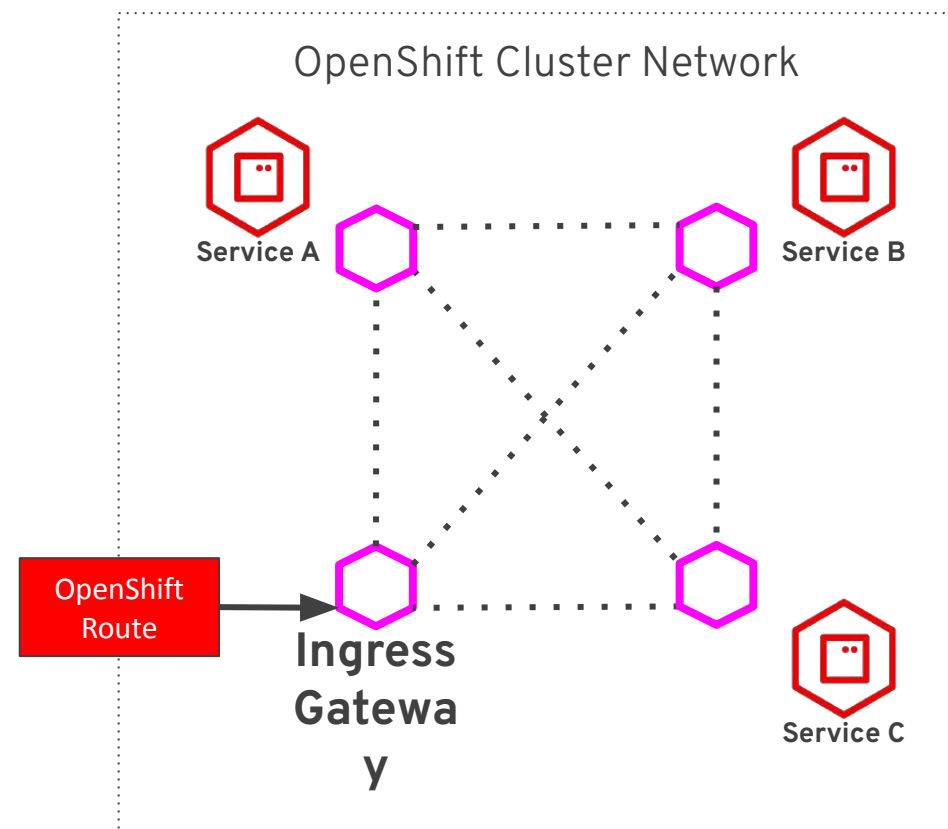
Multi-Tenant

- OpenShift Service Mesh fournit une **topologie multi-tenant**
- Un Service Mesh est constitué d'un ou plusieurs projets (namespaces).
- Chaque Service Mesh est **isolé et géré indépendamment**.
- La communication entre les Service Mesh implique la configuration d'une ou plusieurs **passerelles**, comme vous le feriez pour accéder à des services externes.



Service Mesh avec les OpenShift Routes

- Une **Ingress Gateway** est utilisée pour accéder aux services dans le Service Mesh
- **L'Ingress Gateway** est un **proxy Envoy** autonome qui agit comme un point d'entrée dans le Service Mesh.
- Dans OpenShift, une route agit comme un **point d'entrée** dans le cluster, soutenu par HAProxy.
- OpenShift Service Mesh crée et **configure automatiquement les routes** lors de la création des Ingress Gateways.



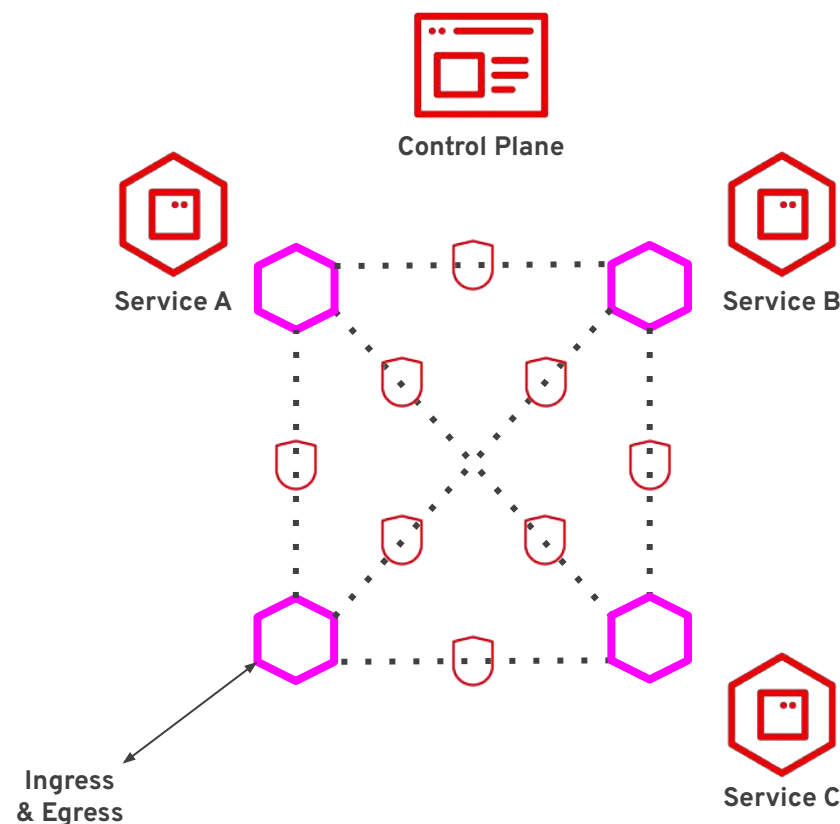
* OpenShift prend également en charge Kubernetes Ingress (qui s'inspire des routes) et Red Hat contribue activement à la prochaine génération d'Ingress - API de service.

Pourquoi utiliser le Service Mesh ?

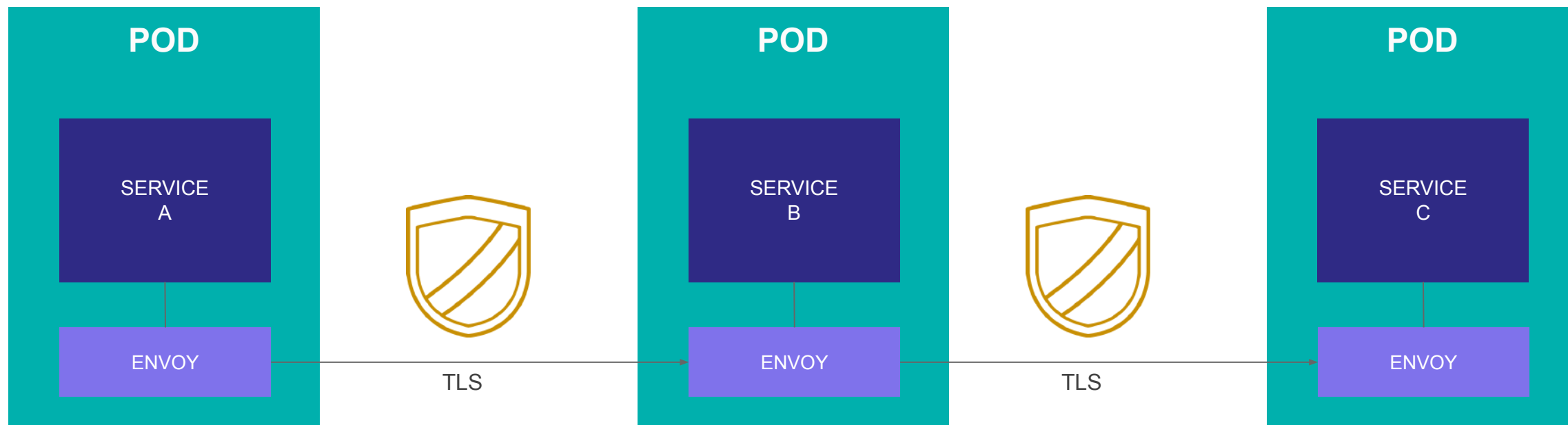
Connecter – Sécuriser – Surveiller – Contrôler

Use Case – Sécuriser les services

- Comme toutes les communications passent par les proxys, nous pouvons **appliquer et gérer les politiques de sécurité entre les services sans modifier le code source.**
 - Appliquer l'utilisation du cryptage mTLS à tous les services.
 - Authentifier les requêtes à l'aide de la validation JSON Web Token (JWT).
 - Définir des politiques d'autorisation « service-to-service » et « user-to-service »
 - Implémentation du « zero-trust networking »
 - Sécuriser le Control Plane du Service Mesh avec des politiques RBAC.

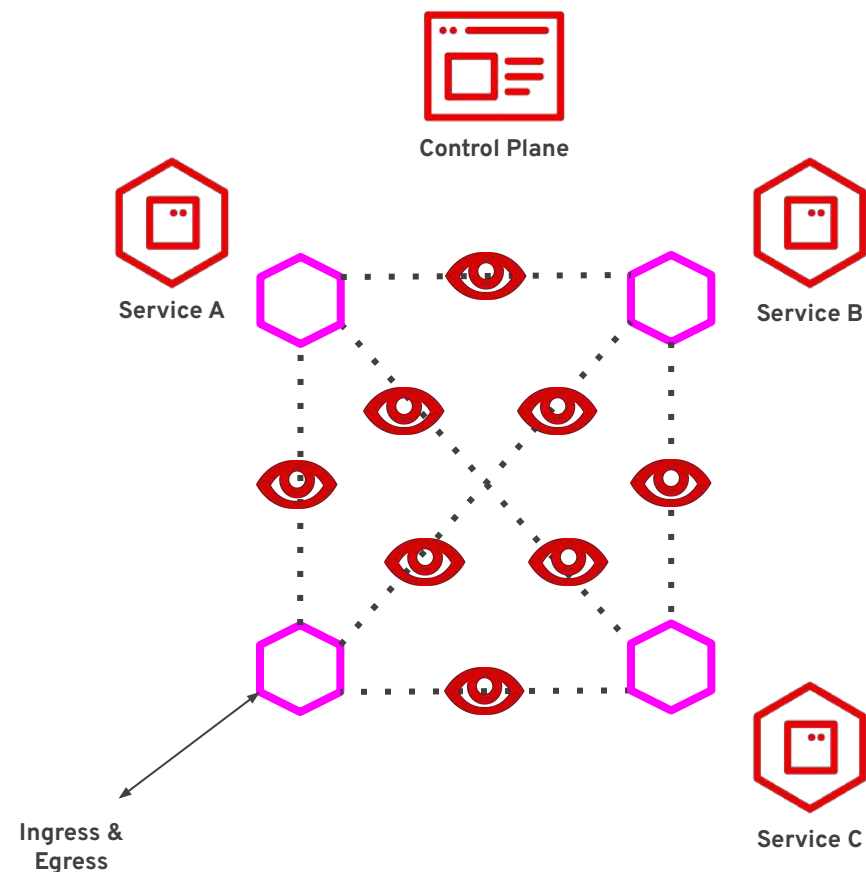


Use Case – Sécuriser les services – Zoom sur le Cryptage mTLS



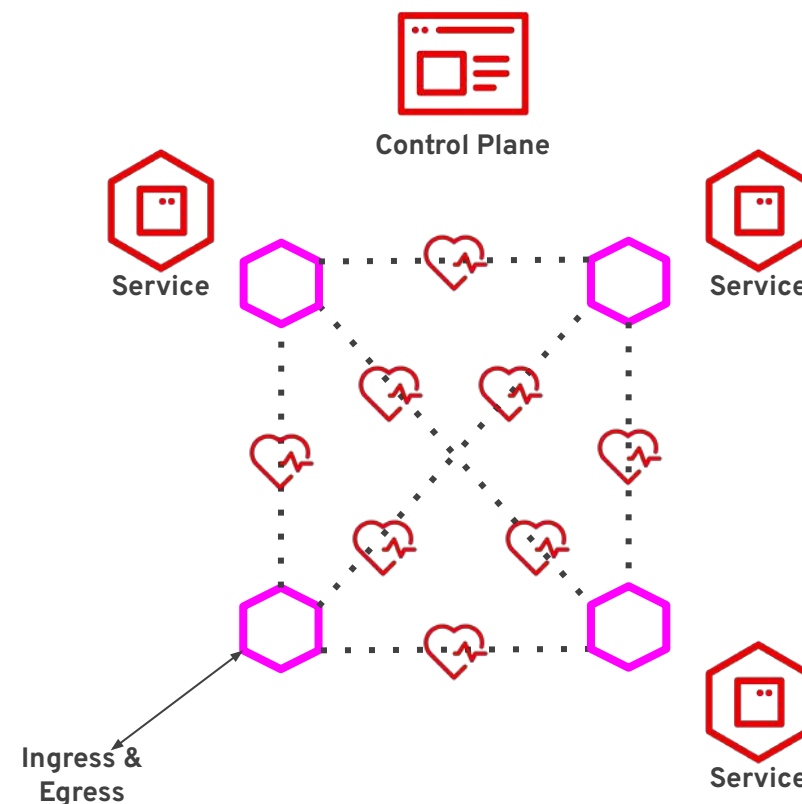
Use Case – Surveiller les services

- Comme la communication se fait via des proxies, nous avons une **visibilité totale de tout le trafic** sans modification du code source*.
 - Métriques et tableaux de bord - volumes de demandes, durée, taux de réussite/échec, etc.
 - Traçage décentralisé - identifiez les goulots d'étranglement provenant de requête lentes.
 - OpenShift Service Mesh comprend Kiali, Jaeger, Grafana, Prometheus, ElasticSearch.



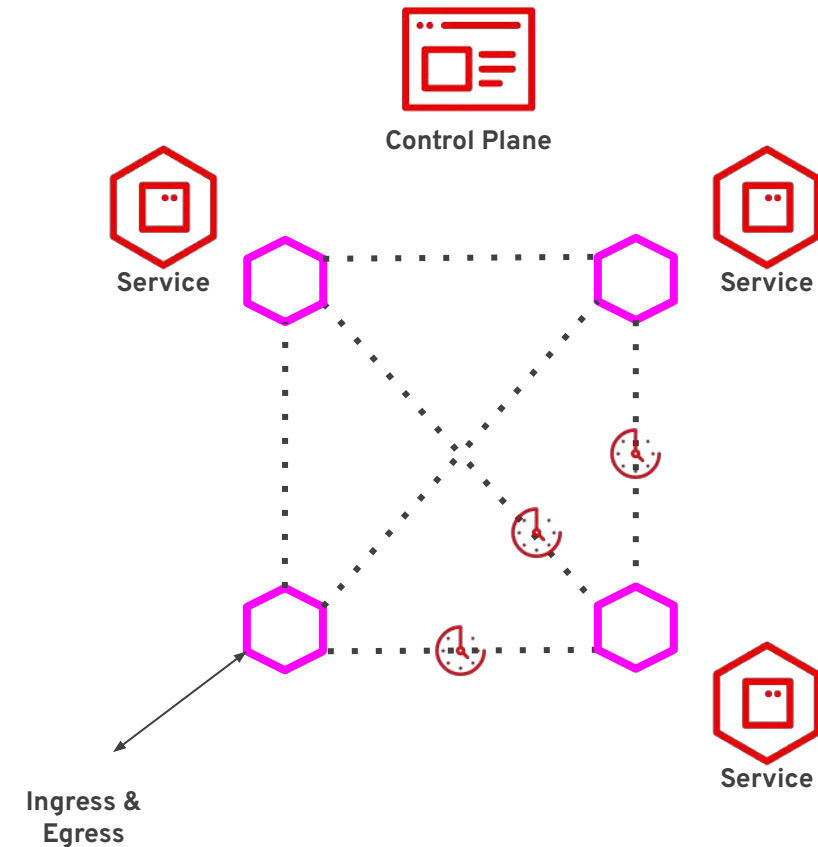
Use Case – Créer des services résilients

- Comme la communication se fait via des proxies, nous pouvons **rendre le système plus résilient sans modifier le code source.**
 - Les « Circuit Breakers » et les « Timeouts » fonctionnent ensemble pour empêcher les défaillances en cascade entre les services.
 - Les « Configurables retries » permettent de surmonter les défaillances à court terme des réseaux et des services.
 - L'injection d'erreurs permet de valider la façon dont les services se comporteront lorsque des défaillances se produiront inévitablement.
 - Permet l'ingénierie du chaos.



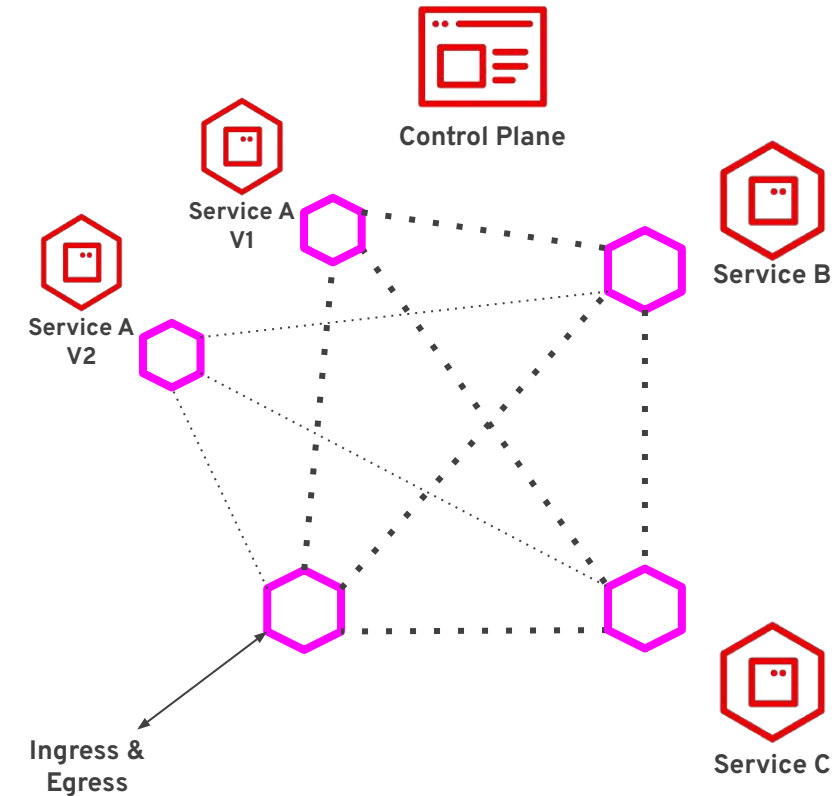
Use Case – Tester la résilience des services

- L'injection d'erreurs permet de valider la façon dont les services se comporteront lorsque des défaillances se produiront inévitablement.
 - Exemples :
 - 50% des demandes à un service échoueront avec le code d'erreur 503.
 - 20% des services seront retardés de 5 secondes.

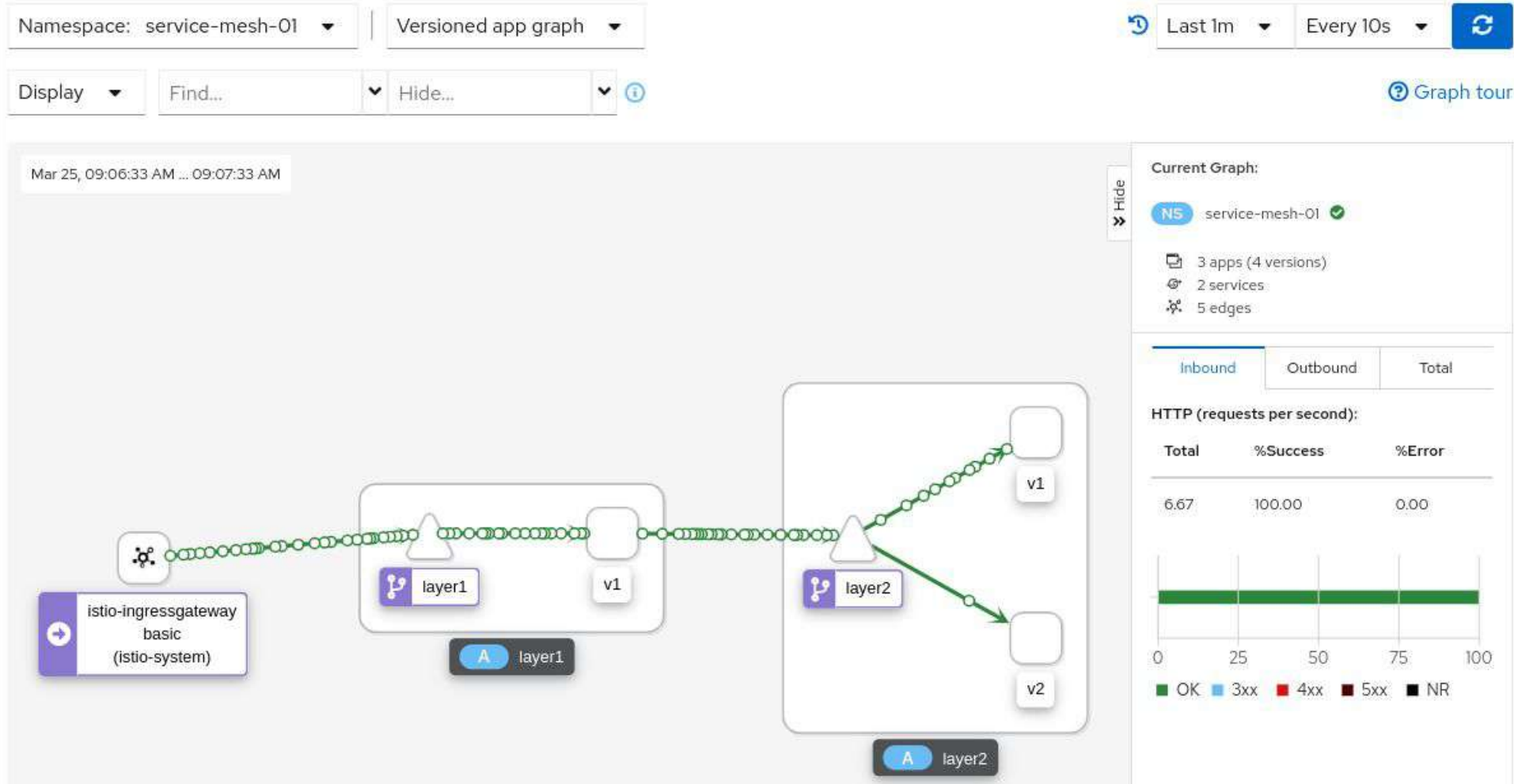


Use Case – Déployer des services

- Le contrôle de toutes les communications permet un **contrôle précis du trafic entre les services sans modification du code source** ni redémarrage des services.
- Créez plusieurs "**sous-ensembles**" d'un service (par exemple, des versions différentes) pour permettre :
 - Canary Deployments
 - A/B Testing
 - Mirror deployments
 - Header based routing



Kiali – Service de Visualisation



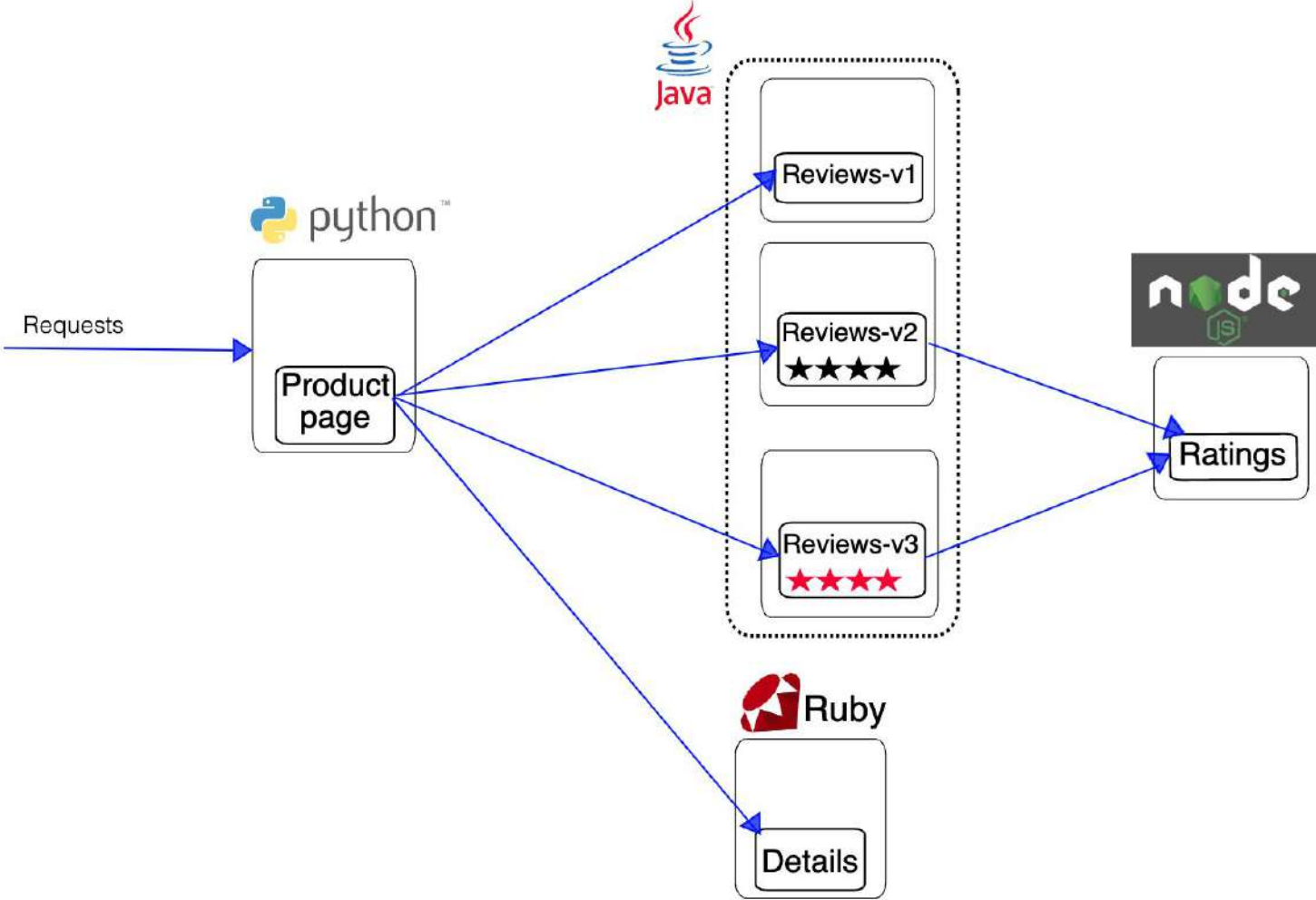
Démo !

Canary Deployment OpenShift Service Mesh

Installation d'OpenShift Service Mesh – Depuis l'Operator Hub

1	OpenShift Elasticsearch Operator
2	Red Hat OpenShift distributed tracing platform
3	Kiali Operator
4	Red Hat OpenShift Service Mesh

Démo – Installation de Bookinfo



Bookinfo Application without Istio

The following ports and protocols are used by the Istio sidecar proxy (Envoy).

 To avoid port conflicts with sidecars, applications should not use any of the ports used by Envoy.

Port	Protocol	Description	Pod-internal only
15000	TCP	Envoy admin port (commands/diagnostics)	Yes
15001	TCP	Envoy outbound	No
15004	HTTP	Debug port	Yes
15006	TCP	Envoy inbound	No
15008	H2	HBONE mTLS tunnel port	No
15009	H2C	HBONE port for secure networks	No
15020	HTTP	Merged Prometheus telemetry from Istio agent, Envoy, and application	No
15021	HTTP	Health checks	No
15053	DNS	DNS port, if capture is enabled	Yes
15090	HTTP	Envoy Prometheus telemetry	No

The following ports and protocols are used by the Istio control plane (istiod).

Port	Protocol	Description	Local host only
443	HTTPS	Webhooks service port	No
8080	HTTP	Debug interface (deprecated, container port only)	No
15010	GRPC	XDS and CA services (Plaintext, only for secure networks)	No
15012	GRPC	XDS and CA services (TLS and mTLS, recommended for production use)	No
15014	HTTP	Control plane monitoring	No
15017	HTTPS	Webhook container port, forwarded from 443	No

Virtual service & Destination rule

```
apiVersion: .../v1alpha3
kind: VirtualService
metadata:
  name: layer2
spec:
  hosts:
  - layer2
  http:
  - route:
    - destination:
        host: layer2
        port:
            number: 8080
        subset: v1
        weight: 80
    - destination:
        host: layer2
        port:
            number: 8080
        subset: v2
        weight: 20
```

```
apiVersion: .../v1alpha3
kind: DestinationRule
metadata:
  name: layer2
spec:
  host: layer2
  subsets:
  - name: v1
    labels:
      Version: v1
  - name: v2
    labels:
      Version: v2
```

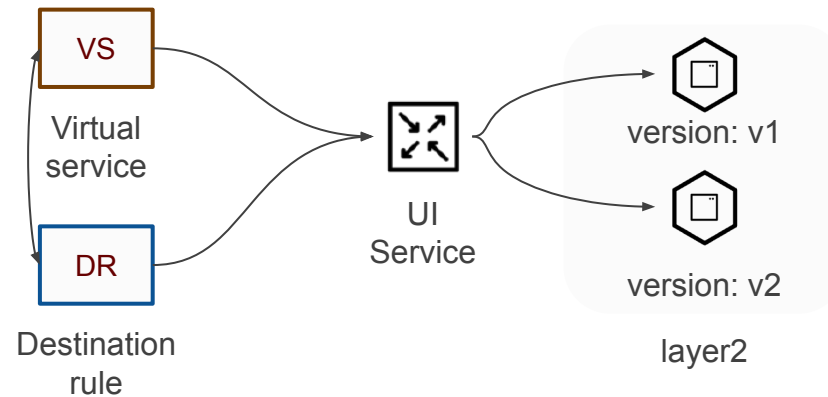
```
apiVersion: v1
kind: Service
metadata:
  name: layer2
labels:
  app: layer2
spec:
  ports:
  - name: http
    port: 8080
selector:
  app: layer2
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: layer2-v1
labels:
  app: layer2
  Version: v1
```

80%

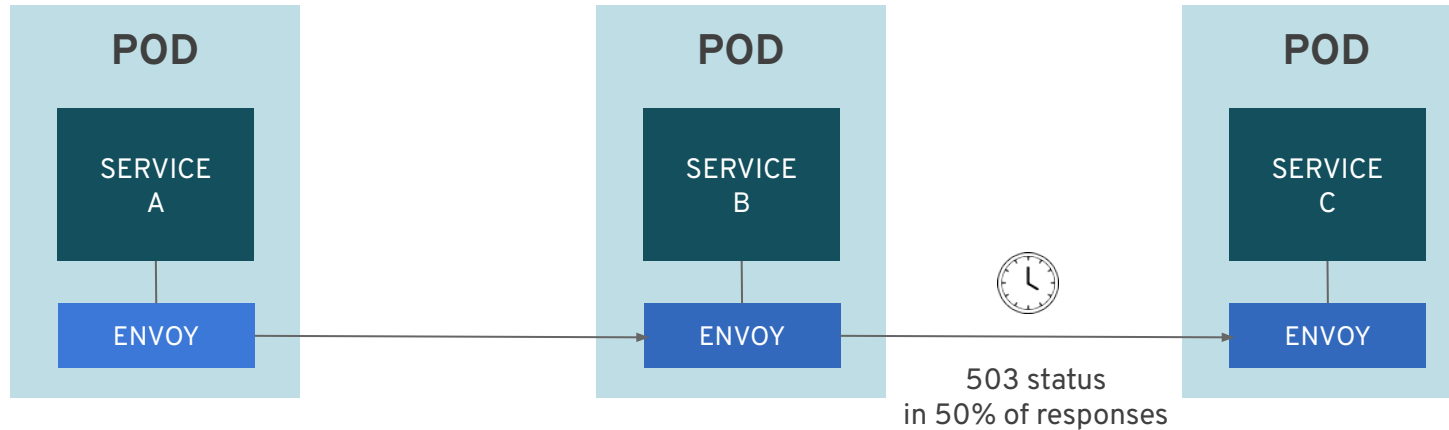
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: layer2-v2
labels:
  app: layer2
  version: v2
```

20%



Tester la résilience des Services

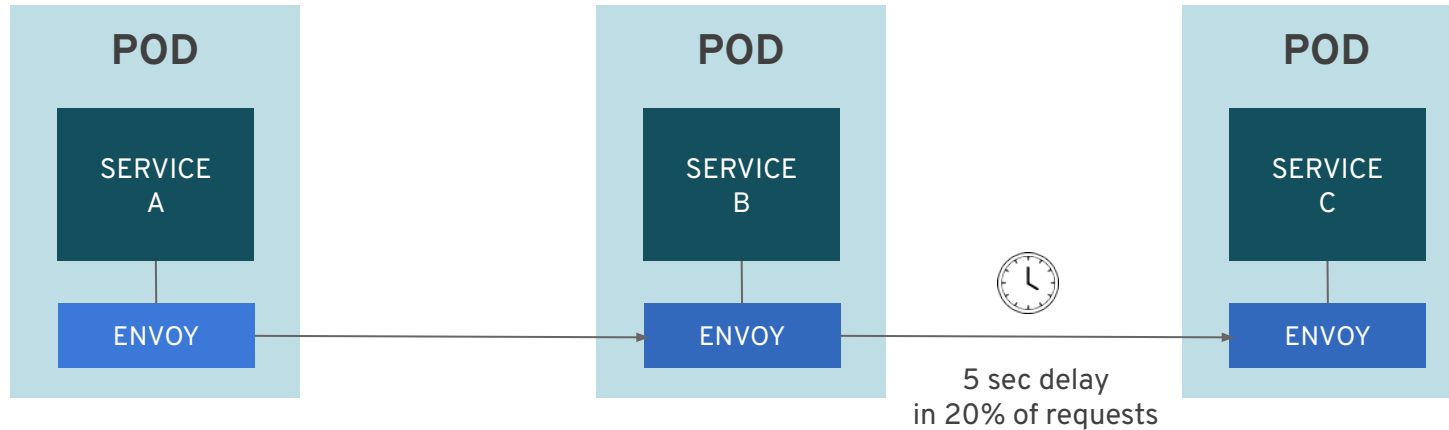
- L'injection d'erreurs permet de valider la façon dont les services se comporteront lorsque des défaillances se produiront inévitablement.
- Exemples :
 - 50% des demandes à un service échoueront avec le code d'erreur 503.



```
apiVersion:
networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: userprofile
spec:
  hosts:
  - userprofile
  http:
  - fault:
      abort:
        httpStatus: 503
        percentage:
          value: 50
    route:
    - destination:
        host: userprofile
        subset: v3
```

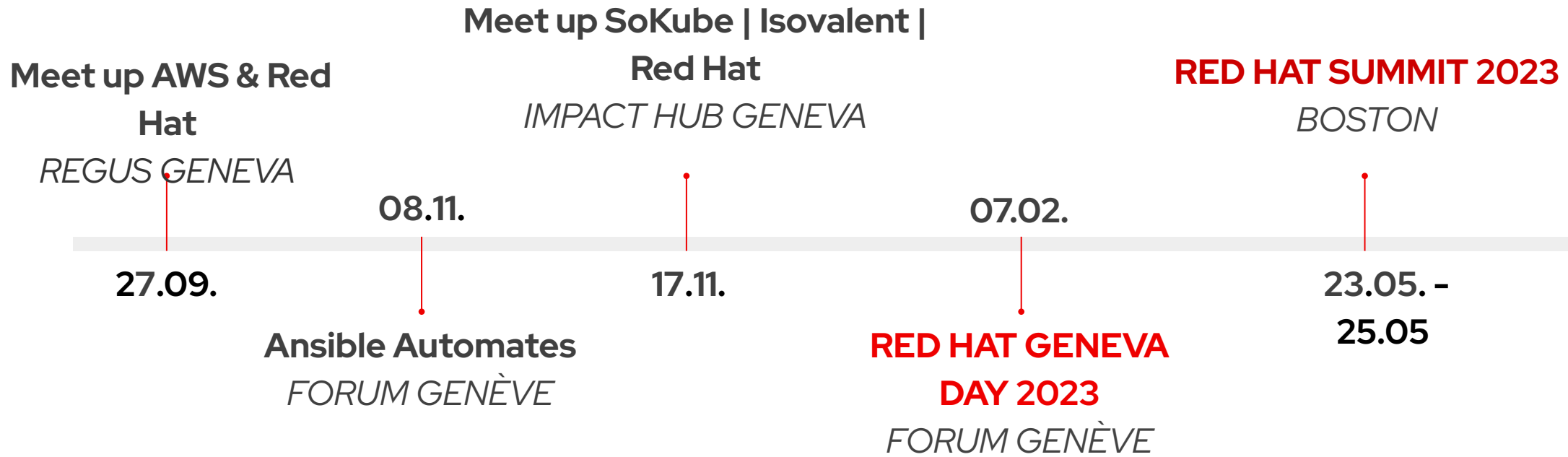
Tester la résilience des Services

- L'injection d'erreurs permet de valider la façon dont les services se comporteront lorsque des défaillances se produiront inévitablement.
 - Exemples :
 - 20% des services seront retardés de 5 secondes.



```
apiVersion:
networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: userprofile
spec:
  hosts:
  - userprofile
  http:
  - fault:
      delay:
        fixedDelay: 5s
        percentage:
          value: 20
    route:
    - destination:
        host: userprofile
        subset: v3
```

Save the dates



SAVE THE DATE

Red Hat
Summit

Connect

Boston, US | May 23-25, 2023

Save the date



SCAN ME



SCAN ME



SCAN ME



Scannez-moi !

Qu'attendez-vous pour vous jeter à *l'eau?*

Frédéric Legrand

CTO - DevOps Transformation Expert Trainer &
Technologist (DevOps, Docker & Kubernetes certified)